A Text Editor

MULTI-EDIT

Ver 3.0

(C) Copyright 1987, 1988, 1989 by *European Cybernetics*

Macro Language Reference Manual

TABLE OF CONTENTS

1. INTRODUCTION 1

3. COMPILING A MACRO 4

4. EXECUTING A MACRO 5

5. LABELS 6

6. COMMENTS 6

7. DATA TYPES 7

8. DEFINING LOCAL VARIABLES 8

9. DATA CONSTANTS 9
    PREDEFINED CONSTANTS 10

10. DEFINING GLOBAL VARIABLES 11
    GLOBAL_STR 11
    GLOBAL_INT 11
    SET_GLOBAL_STR 12
    SET_GLOBAL_INT 12
    FIRST_GLOBAL 12
    NEXT_GLOBAL(int_variable) 12

11. INTEGER EXPRESSIONS 13

12. STRING EXPRESSIONS 14

13. REAL NUMBER EXPRESSIONS 15
    REAL NUMBER OPERATIONS 16
    REAL_I 16
    INT_R 16
    RSTR 16
    RVAL 17

14. UNCONDITIONAL BRANCHING 18
    CALL 18
    RET; 18
    GOTO 18

15. CONDITIONAL BRANCHING 19
    WHILE expression DO 19
    IF expression THEN 19

16. MACRO OPERATIONS 21
    RUN_MACRO 21
    LOAD_MACRO_FILE 21
    UNLOAD_MACRO 22
    INQ_MACRO 22
    FIRST_RUN 22
    FIRST_MACRO 22
    NEXT_MACRO 22

17. PARAMETER PASSING AND RETURN VALUES 23
    MPARM_STR 23
    PARSE_STR 23
    PARSE_INT 23

18. STRING MANIPULATION OPERATIONS 25
    ASCII 25
    CAPS 25
    CHAR 25
    COPY 25
    LENGTH 25
    POS 26
    XPOS 26
    STR_DEL 26
    STR_INS 27
    STR 27
    VAL 27
    REMOVE_SPACE 27
    GET_EXTENSION 27
    GET_PATH 28
    TRUNCATE_EXTENSION 28
    TRUNCATE_PATH 28

19. TEXT MANIPULATION OPERATIONS 29
    CUR_CHAR 29
    GET_WORD 29
    GET_LINE 29
    PUT_LINE 29
    CR 30

TEXT 30
DEL_CHAR 30
DEL_CHARS 30
DEL_LINE 31
BACK_SPACE 31

20. CURSOR POSITION OPERATIONS 32
HOME 32
EOL 32
TOF 32
EOF 32
UP 32
DOWN 32
LEFT 32
RIGHT 33
WORD_LEFT 33
WORD_RIGHT 33
FIRST_WORD 33
MARK_POS 33
GOTO_MARK 34
POP_MARK 34
GOTO_LINE 34
GOTO_COL 34
PAGE_UP 34
PAGE_DOWN 34
AT_EOF 35
AT_EOL 35
C_COL 35
C_LINE 35
C_ROW 35
NEXT_PAGE_BREAK 36
LAST_PAGE_BREAK 36

21. TABSTOPS AND INDENTING 37
TAB_RIGHT 37
TAB_LEFT 37
INDENT 37
UNDENT 37
INDENT_LEVEL 37
SET_INDENT_LEVEL 37
CRUNCH_TABS 38
EXPAND_TABS 38
TABS_TO_SPACES 38

22. BLOCK OPERATIONS 39
  - BLOCK_BEGIN 39
  - COL_BLOCK_BEGIN 39
  - STR_BLOCK_BEGIN 39
  - BLOCK_END 39
  - COPY_BLOCK 39
  - MOVE_BLOCK 39
  - WINDOW_COPY 40
  - WINDOW_MOVE 40
  - DELETE_BLOCK 40
  - BLOCK_STAT 40
  - BLOCK_LINE1 40
  - BLOCK_LINE2 41
  - BLOCK_COL1 41
  - BLOCK_COL2 41
  - MARKING 41
  - BLOCK_OFF 41

23. SCREEN OPERATIONS 42
  - PUT_BOX 42
  - WRITE 43
  - MAKE_MESSAGE 43
  - CLR_LINE 43
  - SCREEN_LENGTH 44
  - SCREEN_WIDTH 44
  - WHEREX 44
  - WHEREY 44
  - GOTOXY 44
  - TOGGLE_VIDEO 44
  - VIDEO_CARD 45
  - VIDEO_MODE 45
  - PUT_LINE_NUM 45
  - PUT_COL_NUM 45
  - WORKING 45
  - SCROLL_BOX_UP 46
  - SCROLL_BOX_DN 46
  - CLEAR_SCREEN 46

24. USER INPUT AND INTERFACE 47
  - READ_KEY 47
  - CHECK_KEY 47
  - KEY1 48

KEY2 48
PUSH_KEY 48
PASS_KEY 48
BAR_MENU 49
V_MENU 50
STRING_IN 50

25. FUNCTION KEY LABEL OPERATIONS 51
PUSH_LABELS 51
POP_LABELS 51
FLABEL 51

26. WINDOW OPERATIONS 52
CREATE_WINDOW 52
SWITCH_WINDOW 52
SIZE_WINDOW 52
DELETE_WINDOW 53
ERASE_WINDOW 53
MODIFY_WINDOW 53
LINK_WINDOW 54
UNLINK_WINDOW 54
LINK_STAT 54
ZOOM 54
REDRAW 54
NEW_SCREEN 54
CUR_WINDOW 54
WIN_X1 55
WIN_Y1 55
WIN_X2 55
WIN_Y2 55
WINDOW_COUNT 55

27. FILE OPERATIONS 56
LOAD_FILE 56
SAVE_FILE 56
FILE_EXISTS 56
FIRST_SAVE 57
EOF_IN_MEM 57
BUFFER_ID 57
TMP_FILE 57
TMP_FILE_NAME 57
FIRST_FILE 57
NEXT_FILE 58

LAST_FILE_NAME 58

28. SEARCH AND REPLACE OPERATIONS 59
  SEARCH_FWD 59
  SEARCH_BWD 59
  REPLACE 60

29. OPERATING SYSTEM INTERFACE 61
  SHELL_TO_DOS 61
  COMSPEC 61
  GET_ENVIRONMENT 61
  REST_DOS_SCREEN 61
  SAVE_DOS_SCREEN 62
  DATE 62
  TIME 62
  ME_PATH 62
  QUIT 63
  PARAM_COUNT 63
  PARAM_STR 63
  CPU 64
  DOS_VERSION 64
  DOS_BACK 64
  DOS_COLOR 64
  CHANGE_DIR 65
  DEL_FILE 65
  WRITE_SOD 65

30. WORD PROCESSING OPERATIONS 66
  WORD_WRAP_LINE 66
  C_PAGE 66
  PG_LINE 66

31. DOS SHELL OPERATIONS 67
  ENTER_DOS_SHELL 67
  EXIT_DOS_SHELL 67
  DIR 67
  DOS_UP 67
  DOS_DOWN 67
  DOS_TOF 68
  DOS_EOF 68
  DOS_PGUP 68
  DOS_PGDN 68
  UPDATE_DIR 68

    DIR_MASK 68
    DIR_ENTRY 68
    DIR_NUM 69
    DIR_TOTAL 69
    DIR_PATH 69
    FILE_MARKED 69
    MARK_FILE 69
    DOS_FILE_ATTR 70

32. KEY ASSIGNMENTS 71
    MACRO_TO_KEY 71
    CMD_TO_KEY 71
    UNASSIGN_KEY 72
    UNASSIGN_ALL_KEYS 72

33. KEYSTROKE MACROS 73
    KEY_IN 73
    PLAY_KEY_MACRO 73
    KEY_RECORD 73
    EXTENDED KEY SYNTAX 74
    COMMANDS THAT ARE DIRECTLY ASSIGNABLE TO A KEY 75

34. MISCELLANEOUS OPERATIONS 76
    BEEP 76
    DELAY 76
    NOT 76
    MODE 77
    HELP 77
    RETRIEVE_HELP 78
    PRINT_STR 78
    VERSION 78
    PUSH_UNDO 79
    POP_UNDO 79
    CALCULATOR 79

35. SYSTEM VARIABLES 80
    STRINGS 80
        RETURN_STR 80
        FILE_NAME 80
        FORMAT_LINE 80
        DEFAULT_FORMAT 80
        PAGE_STR 80
        WORD_DELIMITS 80

SHADOW_CHAR 80
INTEGERS 81
RETURN_INT 81
ERROR_LEVEL 81
IGNORE_CASE 81
REFRESH 81
FILE_CHANGED 81
INSERT_MODE 81
PRINT_MARGIN 81
MESSAGES 81
MOUSE 81
LOGO_SCREEN 82
EXPLOSIONS 82
TRUNCATE_SPACES 82
BACKUPS 82
AUTOSAVE 82
UNDO_STAT 82
FORMAT_STAT 82
WRAP_STAT 82
DOC_MODE 83
MEM_ALLOC 83
RIGHT_MARGIN 83
INDENT_STYLE 83
INS_CURSOR 83
OVR_CURSOR 83
CTRL_HELP 83
MOUSE_H_SENSE 83
MOUSE_V_SENSE 84
WINDOW_ATTR 84
REG_EXP_STAT 84
TAB_EXPAND 84
TEXT_COLOR 84
CHANGE_COLOR 84
BACK_COLOR 84
MENU_COLOR 84
STAT_COLOR 84
ERROR_COLOR 84
SHADOW_COLOR 84
STATUS_ROW 85
MESSAGE_ROW 85
MAX_WINDOW_ROW 85
MIN_WINDOW_ROW 85
NAME_LINE 85

FKEY_ROW 85
DISPLAY_TABS 85
INIT_VIDEO_MODE 85
EMS_STAT 85

36. ERROR HANDLING 86

37. RUN-TIME ERRORS 89
RUN-TIME ERROR CODES 89

38. COMPILER ERRORS 91
COMPILER ERROR CODES 91

39. EXAMPLE MACRO 94

Index 95

1. INTRODUCTION

The Multi-Edit macro language is a compiled programming language similar in syntax to Pascal or Modula-2. It allows the user to customize and add complex, specialized functions to Multi-Edit. Though not as sophisticated as a full fledged structured programming language like C, Pascal or Modula-2, the Multi-Edit macro language does have:

- IF THEN, and WHILE DO constructs.
- String, integer, character and floating point variable types.
- User definable local and global variables.
- Parameter passing and return values
- Complex nested expression evaluation.
- Overlay capabilities.
- Full access to all Multi-Edit functions.
- Easy access to the screen and keyboard.

2. *MACRO STRUCTURE*

A macro source file may contain any number of macros. Each macro is defined as follows:

```
$MACRO name [TO key] [FROM mode] [attributes];
    <body of macro>
END_MACRO;
```

The name of a macro must be a legal DOS file name (without the extension). It may be from 1 to 8 characters long.

Everything in braces ('[]') is optional. 'TO key' defines a key that the macro will be assigned to. 'FROM' mode specifies the Multi-Edit mode that the macro may be invoked from. The currently available modes are: EDIT, DOS_SHELL and ALL. If no 'FROM' is specified, then EDIT is used by default.

The attributes may be any combination of 'TRANS', 'DUMP' and 'PERM'.

'TRANS' specifies a transient macro. A transient macro only brings its code in from disk when it is run. Upon completion, the bulk of it is dumped out of memory, until it is run again. This allows for a great many macros to be loaded, while hogging very little memory.

'DUMP' specifies that the macro is to be completely dumped from memory after it is run.

'PERM' specifies that a macro may NOT be loaded over by another macro of the same name.

NOTES:

A. The Multi-Edit macro language is NOT case sensitive.

B. The assumed file name extension for a macro source file is .SRC.

C. The assumed file name extension for a compiled macro is .MAC.

The following example macro would count the number of mismatched C braces ('{' and '}') in a source file:

```
$MACRO Bracecnt TO <AltB> FROM Edit TRANS;
  DEF_INT( count );                  { Declare integer variable }
  Tof;                               { Goto top of file }
  Count := 0;                        { Initialize variable }
  WHILE Search_Fwd( '@{||@}', 0 ) DO { Look for '{' or '}' }
    IF Cur_Char = '{' THEN           { Determine which char found }
      Count := Count + 1;            { Inc count if char is '{' }
    ELSE
      Count := Count - 1;            { Else Decrement count }
    END;
  END;
                                     { Display result }
  Make_Message( 'Excess { count = ' + Str( Count ) );
END_MACRO;
```

As indicated above, a macro source file may contain any number of macros. When compiled, all the macros in the source file will be included in one .MAC file. Normally the name of the .MAC file that is created during compilation is taken from the name of the first macro in the file. If you wish to give the .MAC file a different name or compile to a specific directory, then you must use the $MACRO_FILE command before any macros are defined in the source file.

EXAMPLE:

```
$MACRO_FILE \PROJ1\SOURCE\PASCAL;

{Pascal indent macro}
$MACRO Pas_Ind;
  <body>...
END_MACRO;

{Pascal BEGIN/END match macro}
$MACRO Pas_Find;
  <body>...
END_MACRO;
```

3. COMPILING A MACRO

A macro is compiled via the 'MEMAC.EXE' macro compiler program supplied with Mult-Edit. The compiler is fully integrated with Multi-Edit. To compile a macro, invoke the COMPILE command from within Multi-Edit (assigned to <CtrlF8> by default, or you can use the Support menu).

When a macro is compiled it's executable (by Multi-Edit) code is automatically saved to disk using the name of the first macro in the source file, with a '.MAC' extension. A $MACRO_FILE statement may be used to override the filename. The .MAC file will be saved to the same directory that the rest of the ME files reside in unless the $MACRO_FILE override is used.

You may also compile a macro from the DOS command line. This is sometimes required due to the memory constraints of some systems. The format for compiling from the command line is:

```
MEMAC <filename> -P<path> -D<debug pc and macro name>
```

The -P option lets you specify the directory that the resulting .MAC file will be written to. The -D debugging option is explained under RUN-TIME ERRORS.

EXAMPLE:

```
MEMAC PASCAL -Pc:\me
```

The above example would compile the file PASCAL.SRC to \ME\PASCAL.MAC.

4. EXECUTING A MACRO

A macro may be executed as follows:

1. The 'Run Macro' function can be invoked and the full name of the macro entered.

2. If the macro has been assigned to a key, then pressing that key (while the editor is in the macros assigned mode) will execute the macro.

3. A macro may be invoked from another macro with the RUN_MACRO statement.

4. A macro may be executed automatically when Multi-Edit is invoked from the DOS command line with the '/R' option. EXAMPLE:

   ```
   C:ME /R SETUP
   ```

   NOTE: The space between the /R and the macro filename is required.

5. When Multi-Edit is started, the macro file STARTUP.MAC is searched for, first in the current directory, and (if not found) in the Multi-Edit directory. If STARTUP.MAC is found, then it is executed. STARTUP.MAC may contain whatever custom initialization code, including loading and running other macros, that is required by the user's specific needs.

5. LABELS

A label is used to define a specific location in a macro program. A label is a string of characters in the set of [a-z,A-Z,_,0-9] with a maximum length of 20 characters, and ending with a ':'. When a label is referenced, the ':' is left off. **NOTE:** A label is local to the macro it is declared in. Identical labels may be declared in different macros that reside in the same source file.

EXAMPLE:

```
$MACRO TEST_MAC;
   {Let's just sit here and loop forever}
   TEST_LABEL:
      GOTO TEST_LABEL;
END_MACRO;
```

6. COMMENTS

Comments may be embedded between curly braces and may be nested.

EXAMPLE:

```
{This line will be ignored by the compiler}

{This is an example of nested commenting.
 The following block of code will be ignored

  {Check the value of JX}
  IF JX <> 0 THEN
     {JX doesn't equal 0 so go ahead}
     Call KILL_LINES;
  ELSE
        {JX = 0 so print error}
     Make_Message( 'ERROR' );
  END:

}
```

7. DATA TYPES

There are four variables types in the macro language: **Integer, String, Character and Real.**

INTEGERS

Integers are always 32 bit and may be any value between -2147483648 and 2147483647.

STRINGS

A string is an array of 8 bit characters. The maximum length of a string is 254 characters. When referencing a character position in a string (with, for example, the COPY operation), the integer value 1 corresponds to the first character in the string. When defined, 255 bytes of memory is immediately allocated for the string.

CHARACTERS

A character or 'Char' occupies a single byte and is fully compatible with string types. Characters are treated as strings of length 1. If you assign a string whose length is greater than 1, to a Char variable, then only the first byte or character of the string will be used. If you assign an empty string (a string whose length is 0) to a char variable, then a value of 0 (ASCII NUL) will be assigned to it.

REALS

The range of a Real number is 1E-38 through 1E+38 with a mantissa of up to 11 significant digits.

BOOLEAN TYPES

There is not a separate BOOLEAN (True or False) type. Instead true or false evaluations and variables return integer results where a non-zero value represents TRUE and a zero value is FALSE.

Type checking is strongly enforced, except between character and string types. For example, attempting to assign an integer to a string would generate a compiler error.

8. DEFINING LOCAL VARIABLES

Local variables are restricted to the scope of the macro they are defined in.

The programmer may define variables of the STRING, INTEGER, CHAR and REAL types with names up to 20 characters long. Variable names follow the same character set convention as labels.
Variables are defined with the following statements:

```
DEF_STR( var_name1, var_name2, ... );

DEF_INT( var_name1, var_name2, ... );

DEF_CHAR( var_name1, var_name2, ... );

DEF_REAL( var_name1, var_name2, ... );
```

EXAMPLES:

```
DEF_STR( Test_String, Output_String, Input_String );
DEF_INT( BCount );
DEF_CHAR( Char1, Char2, Code);
DEF_REAL( RX, RY );
```

You MUST define a variable before using it. If you attempt to use a variable without defining it you will get a compiler error.

9. DATA CONSTANTS

Integer constants are simply entered in their decimal format.

EXAMPLE:

```
Test_Integer := 27;
IF Test_Integer = -346 THEN
   Test_Integer := 346;
END;
```

Hex constants are preceded with a '$'. Note that the maximum number of digits in a hex constant is 4.

Real constants are entered in their decimal format, however they must contain at least one digit left and one digit right of the decimal point.

EXAMPLE:

```
Test_Real := 123.0;
If Test_Real = 0.2343 THEN
   Test_Real := 9.0134;
END;
```

EXAMPLE:

```
Test_Integer := $A9F7;
```

A string constant must be surrounded by single quotes.

EXAMPLE:

```
'This is a string constant'
```

Any ASCII character may be embedded in a string via the '|' character. This is used to embed non-printable or control codes into a string.

EXAMPLES:

'|13' would be a carriage return.
'|12|10' would be a form feed followed by a line feed.
'||' would be a '|' character.

A single quote character (') may be inserted into a string constant by using two single quotes.

NOTE: Currently Multi-Edit does not support user defined named constants (it is coming however).

PREDEFINED CONSTANTS

The following is a list of predefined named constants:

TRUE = 1
FALSE = 0
BLACK = 0
BLUE = 1
GREEN = 2
CYAN = 3
RED = 4
MAGENTA = 5
BROWN = 6
LIGHTGRAY = 7
WHITE = 15

EDIT = 0
DOS_SHELL = 1
ALL = 255
DARKGRAY = 8
LIGHTBLUE = 9
LIGHTGREEN = 10
LIGHTCYAN = 11
LIGHTRED = 12
LIGHTMAGENTA = 13
YELLOW = 14

10. DEFINING GLOBAL VARIABLES

Multi-Edit supports string and integer variables that may be shared between macros. These global variables are not accessed like local variables, but with the following set of procedures and functions.

GLOBAL_STR(Name_String)

RETURNS:	The string value for the global string indicated by Name_string. Note that if the variable has not been set in the editor environment, then an empty string is returned.
EXAMPLE:	Str_Var := GLOBAL_STR('Search_Str');

GLOBAL_INT(Name_string)

RETURNS:	The integer value for the global integer indicated by Name_string. Note that if the variable has not been set in the editor environment, then 0 is returned.
EXAMPLE:	Int_Var := GLOBAL_INT('Common_Line') + 234;

SET_GLOBAL_STR(Name_string, String_value)

ACTION:	Sets the global variable indicated by Name_string to String_value. If the global variable does not yet exist, then it is created.
EXAMPLE:	SET_GLOBAL_STR('Search_Str', 'END;');

SET_GLOBAL_INT(Name_string, Integer_value)

ACTION:	Sets the global variable indicated by Name_string to Integer_value. If the global variable does not yet exist, then it is created.
EXAMPLE:	SET_GLOBAL_INT('Common_Line', 234);

FIRST_GLOBAL(int_variable)

RETURNS:	The name of the first global that is in memory. *Int_Variable* (which must be a user defined local variable) will be set to 1 if the global is an integer, 0 if the global is a string. This initiates calls to NEXT_GLOBAL, allow access to the entire list of all globals in memory.

NEXT_GLOBAL(int_variable)

RETURNS:	The name of the next global in memory. Returns a NUL string if there are no more globals to list. *Int_variable* will be set to 1 if the global is an integer, 0 if it is a string. See the macro source file "GLOBALS.SRC" for an example. The GLOBALS macro creates a list of all of the global variables in memory.

NOTE: Global variable names are not case sensitive.

11. INTEGER EXPRESSIONS

Assignments to integer variables are made as follows:

```
Int_Var := Int_expression;
```

Int_expression may be an integer function, variable or arithmetic expression.

The following arithmetic operators are available for integers:

+	Addition
-	Subtraction
*	Multiplication
/	Division
AND	Logical and bitwise ANDing.
OR	Logical and bitwise ORing.
SHL	Bitwise shift left. EXAMPLE: (JX := JX SHL 4)
SHR	Bitwise shift right.
MOD	Modulus operator.

The following relational operators are available for integers:

=	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
<>	Is NOT equal to

A relational expression returns a 1 for true, and a 0 for false.

EXAMPLE:

```
INT_VAR := 1 > 2;
```

{ INT_VAR would be assigned the value of 0 (FALSE) }

Parentheses may be used to force the order of evaluation.

EXAMPLE:

```
INT_VAR := ((A + B) * C) / D;
```

12. STRING EXPRESSIONS

Assignments to string variables are made as follows:

```
Str_Var := Str_expression;
```

Str_expression may be a string function, variable or expression.

The + operator is used for string concatenation. EXAMPLE:

```
Str_Var := 'This is a test' + Str_Var2 + File_Name;
```

The following relational operators are available for strings:

=	Is equal to
<>	Is NOT equal to

A relational expression returns an integer value of 1 for true, and a 0 for false.

NOTE: String comparisons ARE case sensitive.

EXAMPLE:

```
IF Str_Var = 'ON' THEN
  Make_Message( 'ON');
ELSE
  IF Str_Var <> 'OFF' THEN
    Make_Message( 'UNKNOWN' );
  ELSE
    Make_Message( 'OFF' );
  END;
END;
```

13. REAL NUMBER EXPRESSIONS

Assignments to real variables are made as follows:

Real_Var := Real_expression;

Real_expression may be a real function, variable or arithmetic expression.

The following arithmetic operators are available for reals:

+	Addition
-	Subtraction
*	Multiplication
/	Division

The following relational operators are available for reals:

=	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
<>	Is NOT equal to

Parentheses may be used to force the order of evaluation.

EXAMPLE:

```
REAL_VAR := ((A + B) * C) / D;
```

REAL NUMBER OPERATIONS

REAL_I(int_value)

RETURNS:	A real number that is converted from the passed integer value.
EXAMPLE:	Real_Var := 124.023 + REAL_I(Int_Var);

INT_R(real_value)

RETURNS:	An integer number that is converted from the passed real value.
EXAMPLE:	Int_Var := INT_R(Real_Var) + 12;

RSTR(real_value, int_val1, int_val2)

RETURNS: A string representation of real_value.

Real_value is the real number one wishes to convert to a string. Int_val1 = the number of characters that the resulting string will be left justified in. Int_Val2 = the number digits after the decimal point the string is to contain.

EXAMPLE:

```
    {Convert Test_Real to a string containing 6 significant
     digits and 4 digits after the decimal point. }
Test_Str := RSTR( Test_Real, 6, 4 );
```

RVAL(result_real, string)

ACTION:	Converts a string representation of a number to a real number.
RETURNS:	The integer 0 if no errors occurred due to non numeric characters in the string, or the position of the first non numeric character.

EXAMPLE:

```
Test_Str := '899.25';
   {Convert Test_Str to a real number}
IF RVal( T_Real, Test_Str ) <> 0 THEN
   {Error occurred, inform user}
   Make_Message(
      'ERROR Converting string to real.'
         );
ELSE
   {No Error,  display result of conversion}
   Make_Message(
      'Value is:  '  +  RStr( T_Real )
   );
END;
```

14. UNCONDITIONAL BRANCHING

CALL label;

Runs a subroutine.

RET;

Exits from subroutine. Note that there should be a RET executed for every CALL executed.

GOTO label;

Branches directly to a label.

15. CONDITIONAL BRANCHING

```
WHILE expression DO
   statement;
   statement; ...
END;
```

```
IF expression THEN
   statement;
   statement; ...
END;
```

```
IF expression THEN
   statement;
   statement; ...
ELSE
   statement;
   statement; ...
END;
```

Expressions may be simple or complex evaluations. All evaluations return integer (TRUE = 1 or FALSE = 0) results. The following are the basic evaluation constructs.

```
x = y,        x <> y,x < y,      x > y,        x <= y,
x >= y,       x AND y,           x OR y
```

Examples:

```
 {A useless example of WHILE}
JX := 12;
WHILE JX <= JY DO
  JX := JX + 1;
  IF JX > JY THEN
    Make_Message( 'THIS IS THE LAST TIME!' );
  END;
END;
```

```
IF Our_String = 'This is a test.' THEN
  Make_Message( Our_String );
ELSE
  IF Our_String = 'This is not a test.' THEN
    Make_Message( 'Why isn't it?' );
  END;
END;
```

```
IF ((JX + 1) > 4) AND
      ((Our_String + Your_String) <> 'WHAT?') THEN
  Text('It WORKS!');
END;
```

16. MACRO OPERATIONS

RUN_MACRO(Macro_spec)

ACTION: Executes a macro. Macro_Spec consists of a macro name, an optional macro file name, and an optional parameter string. The format is as follows:

Macro_file_name^Macro_name Parameter_string

Notice the caret "^" delimiter between the macro file name and the macro name, and the space " " delimiter between the macro name and the parameter string. Both the file name and macro name may be any valid DOS file name (including a path). When Multi-Edit executes a macro, it first looks for the macro in memory. If it doesn't find it then it checks the Multi-Edit subdirector (found with the ME_PATH function). If the macro cannot be found, then ERROR_LEVEL will be set to NON-ZERO. The '.MAC' file name extension is assumed.

EXAMPLE:

```
  { Run the LOADFILE macro. If it is not in memory then get
    it from the macro file MEUTIL1.MAC pass a numeric
    parameter /P of 1 and a string paramter /L of "NEW"}
Run_Macro( 'MEUTIL1^LOADFILE /P=1/L=NEW');
```

LOAD_MACRO_FILE(Macro_file_name)

ACTION: Loads a macro file into memory, but does not run it. Check the ERROR_LEVEL flag for errors.

EXAMPLE:

```
  {Load macro file into memory}
Load_Macro_File('Test');
  {If an error occurred, then inform user, else run the macro}
IF Error_Level <> 0 THEN
  Beep;
  Make_Message('Can''t load macro!');
  Error_Level := 0;
ELSE
  Run_Macro( 'Test' );
END;
```

UNLOAD_MACRO(Macro_name)

ACTION: Dumps a macro from memory.

EXAMPLE:
```
   {Dump the macro Test from memory}
Unload_Macro( 'Test' );
```

INQ_MACRO(Macro_name)

RETURNS: True if the macro specified by Macro_name is loaded into memory.

EXAMPLE:
```
   {If macro is not in memory, then load it}
IF Inq_Macro( 'Test' ) THEN
   Load_Macro_File( 'Test' );
   IF Error_Level <> 0 THEN
      Make_Message( 'Error on loading macro.');
      Error_Level := 0;
   END;
END;
```

FIRST_RUN

RETURNS: True (1) if this is the first time a macros been run since it was loaded.

FIRST_MACRO

RETURNS: The name of the first macro that is in memory. This initiates calls to NEXT_MACRO, allow access to the entire list of macros in memory.

NEXT_MACRO

RETURNS: The name of the next macro in list. Returns a NUL string if there are no more macros to list. See the macro source file "MACROS.SRC" for an example. The MACROS macro creates a list of all of the macros in memory.

17. PARAMETER PASSING AND RETURN VALUES

Parameters may be passed to a macro, in text form, in the string parameter passed to the RUN_MACRO procedure. The form is:

```
Run_Macro( 'macro_name parameter_string');
```

The combined length of *macro_name* and *parameter_string* must be <= 254 characters. *Parameter_string* need only be separated from *macro_name* by a space. The *parameter_string* may be accessed by the called macro via the function MPARM_STRING.

Integer values should be returned by the called macro in the system variable RETURN_INT. String values should be returned in RETURN_STR. Remember that RETURN_INT and RETURN_STR are globally accessible, and should only be assigned at the end of a macro.

The functions PARSE_STR and PARSE_INT may be used by the called macro to parse strings and integer values from the passed parameter string.

MPARM_STR

RETURNS: The parameter string passed to the macro.

PARSE_STR(str1, str2)

RETURNS: Returns a string parsed from str2 with str1. Str1 is usually in the form "/name=" where name is any text. Returns a NUL string if str1 was not found in str2. NOTE that this function IS case sensitive. Please see the example on the next page.

PARSE_INT(str1, str2)

RETURNS: An integer value parsed from str2 with str1. Works identically to PARSE_STR except that the string that is parsed is expected to be an numeric string, and is converted to an integer. Please see the example on the next page.

EXAMPLE MACRO USING PARAMETER PASSING:

```
{*************************************************************
 This macro will create a new window and load a specified
 file into it.
 PARAMETERS:        /FN=filename
                    /X1=upper left column of window
                    /Y1=upper left row of window
                    /X2=lower right column of window
                    /Y2=lower right row of window
 RETURNS:  Return_Int = 0, succeeded, <> 0, failed
*************************************************************}
$MACRO GETFILE;
  Def_Str( File_Spec );
  Def_Int( X1, Y1, X2, Y2, Error_Stat );
  Error_Stat := 0;
   {Parse the parameters}
  File_Spec := Parse_Str( '/FN=', MParm_Str );
  X1 := Parse_Int( '/X1=', MParm_Str );
  X2 := Parse_Int( '/X2=', MParm_Str );
  Y1 := Parse_Int( '/Y1=', MParm_Str );
  Y2 := Parse_Int( '/Y2=', MParm_Str );
  Create_Window;
   {If all the size parameters existed (X1 can be left out),
    then Size the window}
  IF (X2 and Y1 and Y2) then
    Size_Window( X1, Y1, X2, Y2 );
  END;
    {If there was a filename parameter then try to load the file}
  IF ( File_Spec <> '' ) THEN
    Load_File( File_Spec );
    Error_Stat := Error_Level;
  END;
  Return_Int := Error_Stat;
END_MACRO;

$MACRO TEST;
  Run_Macro( 'GETFILE /FN=TEST.DOC/X1=1/Y1=3/X2=80/Y2=25' );
  IF (Return_Int <> 0) THEN
    Make_Message( 'Error occurred retrieving file.');
  END;
END_MACRO;
```

18. STRING MANIPULATION OPERATIONS

ASCII(String)

RETURNS:	An integer containing the ASCII value of the first character in the string. If the string is empty, then the result will be 0.
EXAMPLE:	`JX := ASCII( 'A' ); {The ascii value of 'A'}`

CAPS(String)

RETURNS:	The entire string in capitalized form.
EXAMPLE:	`Good_Str := CAPS(Good_Str);`

CHAR(Integer_value)

RETURNS:	A the character (string) equivalent of the passed ASCII value.

COPY(String, Pos, Num)

RETURNS:	A substring (or portion) of String containing Num characters starting at position Pos. If Pos is greater than the length of String, an empty string will be returned. If Pos < 1 or Pos > 254, then a run-time error will occur.

EXAMPLE:

```
        {Get the 1st 3 characters of Temp_Str}
Temp_Str := Copy( Temp_Str, 1, 3 );
```

LENGTH(String)

RETURNS:	The integer length of the string.

EXAMPLE:

```
If Length(Test_String) > 80 Then
        Text('String is too long.');
End;
```

POS(Substring, String)

RETURNS:	An integer containing the position of substring in String. Returns 0 if substring is not found. NOTE that use of the function XPOS is recommended instead. POS has been retained for compatibility with previous versions.
EXAMPLE:	What_Pos := Pos('WHAT', Test_String);

XPOS(Substring, String, Start_position)

RETURNS:	An integer containing the position of substring in String. Returns 0 if substring is not found. The search will start at Start_position in the string. NOTE that XPOS is both faster and more flexible than POS.
EXAMPLE:	What_Pos := XPos('WHAT', Test_String, 12);

STR_DEL(String, Pos, Num)

Returns:	The result of deleting a substring containing Num characters from String starting at position Pos.
EXAMPLE:	Test_String := STR_DEL(Test_String, 9, 4);

STR_INS(Target_string, Destination_string, Location)

RETURNS: The result of inserting the Target_string into the Destination_string at the specified Location.

EXAMPLE:

```
   {Insert 'HELP' into Test_String at position 0}
Test_Str := STR_INS('HELP',Test_String,9);
```

STR(Integer_value)

RETURNS: A string representation of the integer value.

EXAMPLE:

```
   {Put the current line number on the message line}
Make_Message('Processing line: ' + Str(C_Line));
```

VAL(Result, String)

ACTION: Converts a the string expression String to an integer value, stored in Result.

RETURNS: 0 if no errors are detected, otherwise the position in the string of the error is returned.

EXAMPLE:

```
IF Val(Temp_Value, '1000') = 0 Then
   New_Value := Old_Value + Temp_Value;
END;
```

REMOVE_SPACE(String)

RETURNS: The string, with any preceding and trailing spaces removed, and with any occurrences of more than one consecutive space removed.

GET_EXTENSION(Filename)

RETURNS: A string containing the DOS extension part of Filename.

EXAMPLE:

```
Extension := Get_Extension(Old_file);
```

GET_PATH(Filename)

RETURNS: A string containing the DOS path (drive and directory) part of Filename string.

EXAMPLE:

```
If Get_Path(File_Name) <> 'C:\TEST\' Then
   Enter_Dos_Shell;
End;
```

TRUNCATE_EXTENSION(Filename)

RETURNS: A string containing the filename without an extension.

EXAMPLE:

```
New_File :=
   Truncate_Extension(File_Name) + '.NEW';
```

TRUNCATE_PATH(Filename)

RETURNS: A string containing the filename without a path.

EXAMPLE:

```
New_File := 'B:\NEW\' + Truncate_Path(File_Name);
```

19. TEXT MANIPULATION OPERATIONS

CUR_CHAR

RETURNS: The character that the cursor is sitting on in the currently active window. If the cursor is sitting on virtual space or beyond the end of a line then ASCII 255 will be returned.

EXAMPLE:

```
Temp_String := Temp_String + Cur_Char;
```

GET_WORD(Delimit_string)

RETURNS: A portion of the current line starting at the current cursor position and ending before the occurrence of any character in the Delimit string, or the end of the line. Moves the cursor to the end of the returned string.

EXAMPLE:

```
   {Get a word}
New_Word := Get_Word( ' $*().!{}' );
```

GET_LINE

RETURNS: A copy of the entire line the cursor is sitting on.

EXAMPLE:

```
   {Get the current line and put it in Temp_Str}
Temp_Str := Get_Line;
```

PUT_LINE(String)

ACTION: Replaces the line the cursor is sitting on with the string.

EXAMPLE:

```
   {Replace current line with new string}
Put_Line( 'This is a new line.' );
```

CR

ACTION: Performs a carriage return. Will break a line if Insert mode is on. The cursor will be put at the current indent level.

EXAMPLE:

```
   {Insert a new line}
Insert_Mode := TRUE; {Set the insert mode}
EOL;   {goto the end of the line}
CR;    {perform the carriage return}
```

TEXT(String)

ACTION: Inserts (or overwrites) a string of text into the current line at the current cursor position. This operation is sensitive to the insert/overwrite mode.

EXAMPLE:

```
   {Insert some text}
Insert_Mode := TRUE;
Text( 'This is some text.' );
```

DEL_CHAR

ACTION: Deletes the current character.

DEL_CHARS(Count)

ACTION: Deletes Count characters at the current cursor position.

EXAMPLE:

```
Del_Chars( 10 );          {Delete the next 10 chars}
```

DEL_LINE

ACTION: Deletes the current line

EXAMPLE:

```
   {Delete all lines to the end of the file}
WHILE NOT( At_Eof ) DO
   Del_Line;
END;
```

BACK_SPACE

ACTION: Moves the cursor to the left. If Insert_Mode is true, then the current character is deleted, or if the cursor is at column 1, then the current line will be concatenated to the previous line. If Insert_Mode is false (i.e. Overwrite mode) then the character is replaced by a space, and no concatenation of lines will occur.

EXAMPLE:

```
   {Attach current line to previous line}
Insert_Mode := TRUE;
Goto_Col( 1 );          {Goto first character}
Back_Space;             {Perform back space}
```

20. CURSOR POSITION OPERATIONS

HOME

ACTION: Moves the cursor to the currently set indent level.

EOL

ACTION: Moves the cursor to the character position after the end of the current line.

TOF

ACTION: Moves the cursor to the top of the file.

EOF

ACTION: Moves the cursor to the end of the file.

UP

ACTION: Moves the cursor up one line.

DOWN

ACTION: Moves the cursor down one line.

LEFT

ACTION: Moves the cursor to the left one character. If the cursor was at column 1, then the cursor wraps to the end of the previous line.

RIGHT

ACTION: Moves the cursor to the right one character. **NOTE:** The cursor does NOT wrap at the end of the line. The cursor WILL wrap after column 254 is reached.

WORD_LEFT

ACTION: Moves the cursor to the left one word. Will wrap to the previous line. NOTE that the WORD_DELIMITS variable may be used to change the delimiters used to define a word.

WORD_RIGHT

ACTION: Moves the cursor to the right one word. Will wrap to the next line. NOTE that the WORD_DELIMITS variable may be used to change the delimiters used to define a word.

FIRST_WORD

ACTION: Moves the cursor to the first non-white-space character on the line.

MARK_POS

ACTION: Pushes the current cursor position onto the position marker stack for the current window.

EXAMPLE:

```
Mark_Pos;
IF NOT( Search_Fwd( The_Word, 0 ) ) THEN
   Goto_Mark;
END;
```

GOTO_MARK

ACTION:	Restores the cursor position to the last position marked and removes that position from the position marker stack.

POP_MARK

ACTION:	Pulls a mark of the marker stack without actually affecting the cursor.

GOTO_LINE(Line_num)

ACTION:	Moves the cursor to the specified line number. Line_num must be between 1 and 32767.
EXAMPLE:	Goto_Line(C_Line + 10);

GOTO_COL(Column_pos)

ACTION:	Moves the cursor to specified column position. Column_pos must be between 1 and 254.
EXAMPLE:	Goto_Col(10);

PAGE_UP

ACTION:	Moves the cursor one window length up.

PAGE_DOWN

ACTION:	Moves the cursor one window length down.

AT_EOF

RETURNS: True (1) if the cursor is at or beyond the end of the file in the current window.

AT_EOL

RETURNS: True (1) if the cursor is at or beyond the end of the line in the current window.

C_COL

RETURNS: An integer containing the current cursor column position the cursor is on.

EXAMPLE:

```
WHILE C_COL <> 10 DO
  Right;
END;
```

C_LINE

RETURNS: An integer containing the line number the cursor is on in the current file.

EXAMPLE:

```
IF C_Line = 10 THEN
  Down;        {Skip line 10}
END;
```

C_ROW

RETURNS: An integer containing the current row in the window the cursor is on. 1 = the top row currently visible in the window.

EXAMPLE:

```
  {Move cursor to top of window}
WHILE C_Row > 1 DO
  Up;
END;
```

NEXT_PAGE_BREAK

ACTION:	Moves the cursor to the line after the next occurrence of a page break string.

LAST_PAGE_BREAK

ACTION:	Moves the cursor to line after the previous occurrence of a page break string.

21. TABSTOPS AND INDENTING

TAB_RIGHT

ACTION: Moves the cursor one Tabstop to the right. If Insert_Mode = TRUE, then a tab stop

TAB_LEFT

ACTION: Moves the cursor one Tabstop to the left. No characters are inserted or deleted.

INDENT

ACTION: Performs a TAB_RIGHT, then sets the indent level to the current cursor position.

UNDENT

ACTION: Moves the cursor one Tabstop back and sets the indent level to the new position. No characters are inserted or deleted.

INDENT_LEVEL

RETURNS: The column position of the indent level for the current file. This is the position where the cursor will be taken to when a carriage return or a HOME is performed.

SET_INDENT_LEVEL

ACTION: Sets the indent level for the current file to the current column position.

CRUNCH_TABS(string_variable)

ACTION: When GET_LINE is called it returns the line complete with virtual characters (255s). CRUNCH_TABS takes a string variable, and will remove the virtual characters from it. ***String_variable must be*** a user defined string variable.

EXAMPLE:

```
Def_Str( TStr );
TStr := Get_Line;
Crunch_Tabs( TStr );
```

EXPAND_TABS(string_variable)

ACTION: Expands the tabs in the passed string variable to spaces (if the system variable TAB_EXPAND is FALSE) or virtual characters (if TAB_EXPAND is true). *String_variable* must be a user defined variable.

EXAMPLE:

```
Def_Str( Tstr );
Tstr := '|9|9Test paragraph';
Tab_Expand( TStr );
```

TABS_TO_SPACES(string_variable)

ACTION: Converts all tabs and virtual characters in the string variable to space characters.

EXAMPLE:

```
Def_Str( Tstr );
Tstr := Get_Line;
Tabs_To_Spaces( Tstr );
```

22. BLOCK OPERATIONS

BLOCK_BEGIN

ACTION: Marks the start of a block of lines at the cursor.

COL_BLOCK_BEGIN

ACTION: Marks the start of a columnar block of text at the cursor.

STR_BLOCK_BEGIN

ACTION: Marks the start of a stream of text block at the cursor.

BLOCK_END

ACTION: Marks the end of a block of text, either line or column oriented, at the cursor.

COPY_BLOCK

ACTION: Copies the currently marked block of text to the current cursor position.

MOVE_BLOCK

ACTION: Moves the currently marked block of text to the current cursor position.

WINDOW_COPY(window_num)

ACTION:	Copies the currently marked block of text from the specified window into the current window at the current cursor position.

WINDOW_MOVE(window_num)

ACTION:	Moves the currently marked block of text from the specified window to the current window at the current cursor position.

DELETE_BLOCK

ACTION:	Deletes the currently marked block of text.

BLOCK_STAT

RETURNS:	0 if no block is marked. 1 if a line block is marked. 2 if a columnar block is marked. 3 if a stream of text block is marked

BLOCK_LINE1

RETURNS:	The first marked line number.

BLOCK_LINE2

RETURNS:	The second marked line number.

BLOCK_COL1

RETURNS:	The first marked column number.

BLOCK_COL2

RETURNS:	The second marked column number. Will be > 255 if the block is not columnar.

MARKING

RETURNS:	TRUE if block marking is on. Else FALSE.

BLOCK_OFF

ACTION:	Turns the current block off.

23. SCREEN OPERATIONS

PUT_BOX(UL_Col, UL_Row, LR_Col, LR_Row, BG_Color, FG_Color, Title_string, Shadow)

ACTION: Puts a box on the screen of the dimensions specified by UL_Col, UL_Row, LR_Col and LR_Row. BG_Color is the background color, and FG_Color is the foreground color. Title_string specifies a title that will be centered on the top of the box. If Shadow is TRUE, then the box will have a shadow around it. If a shadow is specified, then remember to include the one vertical line and two horizontal spaces that the shadow takes up, in the box dimensions. Note that the section of the screen where the box is drawn is saved onto a stack. It may be restored with the Kill_Box operation.

EXAMPLE:

```
   {Put a box on the screen for error messages}
Put_Box( 5, 4, 62, 12, White, Red, 'ERROR', TRUE );
   {Prompt user}
Write( 'Hit any key to continue...',6,5,White, Red );
   {Wait for key}
Read_Key;
   {Get rid of box}
Kill_Box;
```

KILL_BOX

ACTION: Erases the last created box from the screen and restores the previous screen contents.

EXAMPLE: (see PUT_BOX)

WRITE(String, Col, Row, BG_Color, FG_Color)

ACTION:	Writes a string of text onto the screen at the coordinates specified by Col and Row, with background color BG_Color and foreground color FG_Color. NOTE that this operation does not effect the text in a window in any way, it simply writes to the screen.

MAKE_MESSAGE(message_string)

ACTION:

Writes a message onto the message line of the screen (line 2).

EXAMPLE:

```
   {Display the current character}
Make_Message('The character is: ' + Cur_Char);
```

CLR_LINE(Col, Row, Count)

ACTION:

Clears a section of a line on the screen starting at the Col/Row coordinate and extending Count spaces. Does not effect the text in a file in any way.

EXAMPLE:

```
   {Clear the message line}
Clr_Line( 1, 2, 80 );
   {Message line may also be cleared with:}
Make_Message( '' );
```

SCREEN_LENGTH	
RETURNS:	An integer containing the number of character rows (vertical) displayable on the screen.
SCREEN_WIDTH	
RETURNS:	An integer containing the number of characters (Horizontal) displayable across the screen.
WHEREX	
RETURNS:	The actual x coordinate of the physical on-screen cursor. This has no relationship to a window's virtual cursor position.
WHEREY	
RETURNS:	The actual y coordinate of the physical on-screen cursor. This has no relationship to a windows virtual cursor position.
GOTOXY(x, y)	
ACTION:	Positions the physical cursor at the specified x,y coordinate on the screen. NOTE that this command does not affect the position of a window's virtual cursor.
TOGGLE_VIDEO	
ACTION:	Toggles the current video mode between 25 line mode and whatever denser mode (if any) is supported by the installed video card (EGA = 43 lines, Color 400 = 50 lines).

VIDEO_CARD

RETURNS:

0 = IBM CGA
1 = IBM Monochrome
2 = MONROE Color
3 = MONROE Monochrome
4 = Sigma Color 400
5 = EGA Color
6 = Hercules Monochrome
7 = VGA Monochrome
8 = VGA Color
9 = EGA Monochrome
10 = Wyse 700

VIDEO_MODE

RETURNS:

0 = 80 x 25 (normal)
1 = 80 x 50 (Sigma Color 400)
2 = 80 x 43 (EGA)
3 = 80 x 50 (VGA)

PUT_LINE_NUM(num)

ACTION: Puts the specified number (num is an integer) into the L[ ] display of the current window.

PUT_COL_NUM(num)

ACTION: Puts the specified number (num is an integer) into the C[ ] display of the current window.

WORKING

ACTION: Invokes the flashing 'WORKING' message in the Multi-Edit status line. The message goes away when REDRAW or NEWSCREEN is called.

SCROLL_BOX_UP(x1, y1, x2, y2, attribute)

ACTION:	Scrolls a region of the screen up 1 line. Attribute is a single byte used to set the colors of the new line. The lower 4 bits are the foreground color, the next 3 bits are the background color and the high bit is the blink/underline status.
EXAMPLE:	Scroll_Box_Up(1,1,80,25, (Back_Color shl 4) or Text_Color);

SCROLL_BOX_DN(x1, y1, x2, y2, attribute)

ACTION:	Same as Scroll_Box_Up, except that it scrolls the screen region down.

CLEAR_SCREEN(attribute_byte)

ACTION:	Clears the screen and sets the colors as defined by attribute_byte, where the lower 4 bits are the foreground color, the next 3 bits are the background color, and the high bit is the blink/underline status.

24. USER INPUT AND INTERFACE

READ_KEY

ACTION: Suspends execution and waits for a keystroke. Returns the scan codes of the keystroke in KEY1 and KEY2.

EXAMPLE:

```
  {Get keycode}
Read_Key;
  {If key1 = 0, then extended key was hit}
IF Key1 = 0 AND Key2 = 59 THEN
  Make_Message('You hit F1!');
END;
```

CHECK_KEY

RETURNS: True (1) if a key has been pressed.The scan codes of the keystroke are accessed via the functions KEY1 and KEY2. The equivalent of READ_KEY, without suspending execution.

EXAMPLE:

```
  {If a key has been pressed, then inform user
   of the type of key}
IF Check_Key THEN
  IF Key1 = 0 THEN
    Make_Message('You hit an extended key.');
  ELSE
    Make_Message('You hit a normal key.');
  END:
END;
```

KEY1

RETURNS: An integer containing the first code of the last key that was pressed. A value of zero indicates that an extended key was pressed.

EXAMPLE:

```
  {Check for extended key code}
IF Key1 = 0 THEN
  Key_Code := Key2;
END;
```

KEY2

RETURNS: An integer containing the second code of the last key that was pressed.

EXAMPLE:

```
  {If F1 was pressed, then goto end of macro}
IF Key1 = 0 AND Key2 = 94 Then
  GOTO End_Of_Mac;
END;
```

PUSH_KEY(keycode, scancode)

ACTION: Pushes the passed keycode/scancode combination back onto the Multi-Edit keycode buffer. Up to 16 keycodes may be pushed back.

EXAMPLE:

```
 {Filter out NON extended keys}
Read_Key;
IF (Key1 = 0) THEN
 Push_Key( Key1, Key2 );
END;
```

PASS_KEY(keycode, scancode)

ACTION: Passes the keycode/scancode through to Multi-Edit, which then performs whatever command or macro is assigned to it. This operation is used by the REPEAT macro.

BAR_MENU(Menu_string, Col, Row, Start_num)

ACTION: Creates a horizontal bar menu.

Menu_string is a string constant or variable formatted as follows:
'Item1(h1)Item2(h2)Item3(h3)...'
where Item1 . . Item3 represent menu entries and h1 . . h3 are 2 character indexes for the help file. The first uppercase letter of each item will be highlighted on the screen and used for single key selection (see the example below). Col and Row represent the physical screen coordinates where the menu will be located. Start_num is the number of the menu item that will be highlighted first.

RETURNS: An integer that is the number of the menu item selected, or 0 if the <ESC> key was pressed to exit the menu without making a selection.

EXAMPLE:

```
   {Setup a bar menu}
Result := BAR_MENU('Load(FL)Save(FL)Run(RN)
                                   Compile(CP)',2,2,1);
   {If escape key was pressed, then exit}
IF Result = 0 THEN
   GOTO Exit;
END;
   {If 'Load' was selected, then load the file}
IF Result = 1 THEN
   Load_File('TEST.SRC');
END;
   {If 'Save' was selected, then save the file}
IF Result = 2 THEN
   Save_File;
END; ........
```

V_MENU(String1, String2, String3, Col, Row, Start_num)

ACTION: V_MENU is identical to BAR_MENU (see above) except that the menu is placed on the screen with the items stacked vertically instead of horizontally, and there are three menu strings instead of one. This allows very large vertical menus (as the maximum string length is 254 characters).

STRING_IN(Input string, Prompt string,Length, Col, Row, Help string)

ACTION: Prompts the user to input a string.

Prompt string is a string expression to serve as a prompt to the user. Input string is the string variable where the user response will be stored (one caution is to be sure and initialize Input string either to a default response, or to an empty string). Length is the maximum number of characters the user may input. Help string is the 2 character index into the help file (if <F1> is pressed).

RETURNS: An integer containing 1 (TRUE) if the string was inputted and enter was pressed or 0 (FALSE) if <ESC> was pressed.

EXAMPLE:

```
  {Prompt user for his name}
JX := String_In( Name, 'Enter your name', 20, 2, 2, 'NM');
  {IF escape was pressed then exit}
IF JX = 0 THEN
  GOTO EXIT;
END;
```

25. FUNCTION KEY LABEL OPERATIONS

PUSH_LABELS

ACTION: Pushes a new blank set of function key labels onto the key label stack.

EXAMPLE: (See FLABEL below.)

POP_LABELS

ACTION: Pops the current set of function key labels off the stack, restoring the previous set.

EXAMPLE: (See FLABEL below.)

FLABEL(String, Key_num, Mode)

ACTION: Modifies a key label. String is a six character string variable or literal. Key_num represents the function key number. It is assigned as follows: <F1> through <F10> = 1 through 10 <ShftF1> through <ShftF10> = 11 through 20 <CtrlF1> through <CtrlF10> = 21 through 30 <AltF1> through <AltF10> = 31 through 40 All other numbers are ignored. Mode is the Multi-Edit mode where the label will be used. 0 = EDIT, 1 = DOS_SHELL, 255 = the current labels, regardless of mode.

EXAMPLE:

```
   {Assign label to <F3> from EDIT mode (using the constant 'EDIT')}
FLABEL( 'SAVE', 3, EDIT );
   {Create new set of labels}
PUSH_LABELS;
   {Assign label to <ShftF5> within current labels}
FLABEL( 'RUN_DS', 15, 255 );
   {Wait for key to be pressed}
READ_KEY;
   {Restore old labels}
POP_LABELS;
```

26. WINDOW OPERATIONS

CREATE_WINDOW

ACTION: Creates a new editing window. If more than 100 windows are created, or Multi-Edit cannot allocate enough memory for a new window, then ERROR_LEVEL will be non-zero. (See ERROR HANDLING)

SWITCH_WINDOW(Win_Num)

ACTION: Switches to the window number denoted by the integer Win_Num. If Win_Num is larger than the number of active windows, then it will wrap to window 1.

EXAMPLE:

```
  {Switch to second editing window}
Switch_Window(2);
```

SIZE_WINDOW(UL_Col, UL_Row, LR_Col, LR_Row)

ACTION: Changes the dimensions of the current window. The dimensions are limited by the current Length and Width of the screen, and will be truncated if those values are exceeded. (See SCREEN_WIDTH and SCREEN_LENGTH)

EXAMPLE:

```
  {Change the dimensions of the current window}
Size window(3,1,23,80);
```

DELETE_WINDOW

ACTION: Completely deletes the current window. NOTE THAT ALL DATA IN THE WINDOW WILL BE LOST!

EXAMPLE:

```
   {If no changes detected in the current
    file then delete the current window}
IF File_Changed = 0 THEN
   Delete_Window;
END;
```

ERASE_WINDOW

ACTION: Erases the contents of the window, but does not delete the window itself. NOTE THAT ALL DATA IN THE WINDOW WILL BE LOST!

EXAMPLE:

```
   {If no changes detected in the current file
    then erase the current window}
IF File_Changed = 0 THEN
   Erase_Window;
END;
```

MODIFY_WINDOW

ACTION: Enters the user-oriented 'modify window' routine. Prompts user for the upper left and lower right corners of the new window size.

LINK_WINDOW(Win_Num)

ACTION: Links the current window to the window identified by Win_Num;

EXAMPLE:

```
  {Link the current window to the next window}
Link_Window( Cur_Window + 1 );
```

UNLINK_WINDOW

ACTION: Unlinks the current window.

LINK_STAT

RETURNS: The number of windows - 1 that the current buffer is linked to. 0 if the buffer is linked to only one buffer.

ZOOM

ACTION: Toggles the current window between it's defined size and full screen size.

REDRAW

ACTION: Updates the display of the current window only.

NEW_SCREEN

ACTION: Redraws the entire screen.

CUR_WINDOW

RETURNS: An integer containing the current window number.

EXAMPLE:

```
  {Switch to next window}
Switch_Window( Cur_Window + 1 );
```

WIN_X1

RETURNS:	The upper left hand column of the current window. If it is set to zero, then the left border will not be displayed. The maximum value is limited to the width of the screen minus 2.

WIN_Y1

RETURNS:	The upper left hand row of the current window The minimum value is 3. The maximum value is the length of the screen minus 3.

WIN_X2

RETURNS:	The lower right hand column of the current window. The minimum value is 4. The maximum value is the width of the screen plus 1 (will cause the right border not to be displayed).

WIN_Y2

RETURNS:	The lower right hand row of the current window. The minimum value is 5. The maximum value is the length of the screen minus 1.

WINDOW_COUNT

RETURNS:	An integer containing the number of windows currently active.

EXAMPLE:

```
  {Switch to the very last window}
Switch_Window( Window_Count );
```

27. FILE OPERATIONS

LOAD_FILE(Filename)

ACTION: Loads the specified file into the current window. Get error code via the ERROR_LEVEL variable. (See ERROR HANDLING)

EXAMPLE:

```
   {Load the file}
Load_File( ’Test.Txt’ );
   {If error, then run the error macro}
IF Error_Level <> 0 THEN
   Run_Macro( ’ME-ERROR’ );
END;
```

SAVE_FILE

ACTION: Saves the file in the current window. Get error code via the ERROR_LEVEL variable. (See ERROR HANDLING)

EXAMPLE:

```
   {Save the current file}
Save_File;
   {If error, then run the error macro}
IF Error_Level <> 0 THEN
   Run_Macro( ’ME-ERROR’ );
END;
```

FILE_EXISTS(filename)

RETURNS: True (1) if the specified file exists.

EXAMPLE:

```
   {If file exists then load it}
FStr := ’TEST.SRC’;
If File_Exists( FStr ) THEN
   Load_File( Fstr );
END;
   {If error, then run the error macro}
IF Error_Level <> 0 THEN
   Run_Macro( ’ME-ERROR’ );
END;
```

FIRST_SAVE

RETURNS: True(1) if the current file has been save at least once since the start of the editing session.

EOF_IN_MEM

RETURNS: True (1) if the end of the file is in memory, false (0) if the EOF is still on disk.

BUFFER_ID

RETURNS: The ID # of the file buffer that the current window is attached to.

TMP_FILE

RETURNS: True if a temporary file is being used for this buffer.

TMP_FILE_NAME

RETURNS: The name of the temporary file being used by this buffer.

FIRST_FILE(filename)

ACTION: Searches for the first occurrence of the specified filename. The filename may include DOS wildcard characters (* and ?) and a complete path.

RETURNS: Zero if the file was found, else DOS error code 18 is returned.

EXAMPLE: (see NEXT_FILE)

NEXT_FILE

ACTION: Searches for the next occurrence of the filename started by FIRST_FILE.

RETURNS: Zero if the file was found, else DOS error code 18 is returned.

EXAMPLE:

```
 {Find all occurrences of a filename string and load them}
TStr := '*.SRC';
IF FIRST_FILE(TSTR) = 0 THEN
   TStr := LAST_FILE_NAME;
   Call Do_Load_File;
   WHILE (NEXT_FILE = 0) DO
      TStr := TStr2 + LAST_FILE_NAME;
            {call a routine to load the file}
      Call Do_Load_File;
   END;
END;
```

LAST_FILE_NAME

RETURNS: The last filename found by a FIRST_FILE or NEXT_FILE search.

EXAMPLE: (See NEXT_FILE)

28. SEARCH AND REPLACE OPERATIONS

NOTE: See the Multi-Edit user's guide for information on regular expressions.

SEARCH_FWD(Search_string, Num_lines)

ACTION:	Searches forward for Search_string from the current cursor position Num_lines number of lines. If Num_Lines = 0, then the search is performed to the end of the file. If Search_string is found, then the cursor will be positioned at the first character of the found item. NOTE that the system variable IGNORE_CASE is used to determine whether or not the search is case sensitive.
RETURNS:	1 if the string was found, 0 if not found.
EXAMPLE:	(See REPLACE)

SEARCH_BWD(Search_string, Num_lines)

ACTION:	Searches backward for Search_string from the current cursor position Num_lines number of lines. If Num_lines = 0, then the search is performed to the top of the file. NOTE that the system variable IGNORE_CASE is used to determine whether or not the search is case sensitive.
RETURNS:	An integer. 1 (TRUE) if the string was found, 0 if not found.

REPLACE(replace_string)

ACTION: The last Search match (see SEARCH) is replaced with replace_string. Does not effect the cursor position. NOTE that the cursor must not have moved since the Search.

EXAMPLE:

```
   {Find all occurrences of 'WHAT' or 'WHERE' and
         replace them with 'WHY', ignoring the case}
Ignore_Case := True;
WHILE Search_Fwd('{WHAT}||{WHERE}', 0 ) DO
   Replace( 'WHY' );
END;
```

29. OPERATING SYSTEM INTERFACE

SHELL_TO_DOS(Cmd_string, Cmd_flag)

ACTION: Shells to DOS and executes the command specified in Cmd_string. If Cmd_flag is true then COMMAND.COM is used to execute Cmd_string, else Mult-Edit attempts to run Cmd_string directly. In order to execute any of the internal dos commands like DIR, DEL or COPY Cmd_flag must be true. If Cmd_string = '' (empty string) then COMMAND.COM is simply executed.

NOTE that unlike Multi-Edit DOS Shell procedures and functions, this procedure may be executed from any Multi-Edit mode.

EXAMPLE:

```
  {Copy a file}
SHELL_TO_DOS( 'COPY ME.HLP A:', TRUE );
```

COMSPEC

RETURNS: A string containing the path to find COMMAND.COM.

EXAMPLE:

```
  {Execute COMMAND.COM}
SHELL_TO_DOS( Comspec, FALSE );
```

GET_ENVIRONMENT(entry_name)

RETURNS: Parses the DOS environment area for entry_name and returns the string associated with it.

EXAMPLE:

```
Tstr := Get_Environment( 'ME_PATH=' );
```

REST_DOS_SCREEN

ACTION: Restores the screen that was saved upon startup of Multi-Edit or upon the last call to SAVE_DOS_SCREEN.

EXAMPLE: (See SAVE_DOS_SCREEN)

SAVE_DOS_SCREEN

ACTION: Saves the current screen.

EXAMPLE:

```
   {Restore the previous screen}
RESTORE_DOS_SCREEN;
   {Execute batch file}
SHELL_TO_DOS( 'MAKE_LST', TRUE );
   {Save the screen}
SAVE_DOS_SCREEN;
```

DATE

RETURNS: A string containing the current date according to the system clock. The format is: mm/dd/yy

EXAMPLE:

```
   {Insert the date}
Text('The date of this document is:' + Date);
```

TIME

RETURNS: A string containing the current time according to the system clock. The format is: hh:mm:ssam (or pm)

EXAMPLE:

```
   {Act like an alarm clock}
IF Time = '07:35:00am' THEN
   Beep;
   Make_Message('Good morning Mr. Phelps!');
END;
```

ME_PATH

RETURNS: A string containing the DOS path to find the ME overlay files.

QUIT(Return_Code)

ACTION: Exits Multi-Edit. Return_Code is an integer value, and may be evaluated from a batch file with the DOS ERRORLEVEL operation. Note that if changes have been made to files, and the files have not been saved, then data will be lost.

EXAMPLE:

```
   {Save off current file}
Save_File;
   {If no errors, then quit}
IF Error_Level = 0 THEN
   Quit;
END;
```

PARAM_COUNT

RETURNS: An integer containing the number of parameters passed to Multi-Edit on the dos command line.

EXAMPLE: (See PARAM_STR)

PARAM_STR(param_number)

RETURNS: A string containing the specified command line parameter. With DOS Version 3 and above, a param_number value of 0 will return the name of the command that invoked Multi-Edit. DOS Version 2 will return an empty string for param_number 0.

EXAMPLE:

```
   {Get the first parameter on the command line
    if the parameter was entered.}
IF Param_Count > 0 THEN
   Test_Parameter := Param_Str(1);
END;
```

CPU

RETURNS:

0 = 8086/8088
1 = 80186/80188
2 = 80286
3 = 80386

DOS_VERSION

RETURNS: Integer containing the DOS version number as follows:

Lo Byte = Major version number.
Hi Byte = Minor version number.

EXAMPLE:

```
        {Get the dos version and display it}
JX := DOS_VERSION;
JL := JX AND $00FF;
JH := JX SHR 8;
Make_Message('DOS Version = ' + Str(JL) + '.' + Str(JH));

{If jx equals 2003 hex, then the dos version number would be 3.20}
```

DOS_BACK

RETURNS: The background color (integer) that was being used by DOS before entering Multi-Edit.

DOS_COLOR

RETURNS: The foreground color (integer that was being used by DOS before entering Multi-Edit.

CHANGE_DIR(path_string)

ACTION: Changes the current directory. The DOS DIRECTORY SHELL does NOT need to be active. Error_Level will be non-zero if an error occurred.

EXAMPLE:

```
Change_Dir( '\DOCUMENT' );
IF Error_Level <> 0 THEN
  Make_Message( 'Error changing directory' );
  Error_Level := 0;
END;
```

DEL_FILE(filename_string)

ACTION: Deletes the specified file. Wildcard characters are NOT allowed. The DOS DIRECTORY SHELL does NOT need to be active. Error_Level will be non-zero if an error occurred.

WRITE_SOD(string)

ACTION: Writes the string to the standard output device.

EXAMPLE:

```
 {Write message + CR/LF to standard output}
Write_SOD( 'Lots of FUN!|13|10' );
```

30. WORD PROCESSING OPERATIONS

WORD_WRAP_LINE

ACTION: Starts the wordwrap process from the current line. Use this to reformat a line or paragraph.

C_PAGE

RETURNS: The current page number, starting with 1. (Use in Document mode)

PG_LINE

RETURNS: A line number that is not the actual line number in the file, but rather an offset from the current page. (Use in Document mode)

31. DOS SHELL OPERATIONS

NOTE that DOS SHELL procedures and functions (other than ENTER_DOS_SHELL) can only be run while in DOS SHELL directory mode.

ENTER_DOS_SHELL

ACTION: Enters the DOS Shell mode. Must be invoked from EDIT mode.

EXIT_DOS_SHELL

ACTION: Exits the DOS Shell mode.

DIR(dir_string)

ACTION: Displays the files according to the dir string.

EXAMPLE:

```
   {Go into the DOS SHELL}
Enter_Dos_Shell;
   {Get a directory}
Dir( C:\TEST\*.* );
```

DOS_UP

ACTION: Moves the DOS Shell cursor up one file name.

DOS_DOWN

ACTION: Moves the DOS Shell cursor down one file name.

DOS_TOF

ACTION:	Move directory cursor to the top of the directory list.

DOS_EOF

ACTION:	Move directory cursor to the end of the directory list.

DOS_PGUP

ACTION:	Move directory cursor one page up.

DOS_PGDN

ACTION:	Move directory cursor one page down.

UPDATE_DIR

ACTION:	Cause the directory shell to update the directory list.

DIR_MASK

RETURNS:	A string containing the full path name of the directory (including any wild cards) that the DOS SHELL directory is viewing.

DIR_ENTRY

RETURNS:	A string containing the current file name that the cursor is on in the directory list.

DIR_NUM

RETURNS: The file number (in the directory list) that the cursor is on.

DIR_TOTAL

RETURNS: The total number of entries in the current directory list, including the '.' and '..' directories.

DIR_PATH

RETURNS: A string containing the current directory path.

EXAMPLE:

```
IF Dir_Path = 'A:\TEST\*.*' THEN
   Dir('A:\TEST\TEXT\*.TXT');
END;
```

FILE_MARKED

RETURNS: True if the file that the cursor is on is marked.

MARK_FILE

ACTION: Toggles the marked status of the file that the DOS cursor is on.

DOS_FILE_ATTR

RETURNS:	The integer file attribute for the directory entry the DIRECTORY SHELL cursor is sitting on. Only valid when the DIRECTORY SHELL is active.

Attributes:

$01 = Read Only
$02 = Hidden
$04 = System file
$08 = Volume ID
$10 = Directory
$20 = Archive

32. KEY ASSIGNMENTS

MACRO_TO_KEY(Key_code, Macro_spec, Mode)

ACTION: Assigns a macro to a specified key from a specified Multi-Edit mode. When the key is hit Multi-Edit will attempt to run the macro just as if the Run_Macro function had been invoke.

Key_code is any legal key code constant (see 'EXTENDED KEY SYNTAX' for a complete list of valid key code constants). Macro_spec is a string which should contain a valid macro name, an optional macro file name, and an optional parameter string. The format should be as follows:

Macro_file_name^Macro_name Parameter_String

Notice the caret "^" delimiter between the macro file name and the macro name, and the space delimiter " " between the macro name and the parameter string. Mode is the Mult-Edit mode from which the macro is to be invoked. Currently EDIT, DOS_SHELL and ALL are supported.

EXAMPLE:

```
MACRO_TO_KEY( <AltF9>, 'RUNPROG', Edit);
```

CMD_TO_KEY(Key_code, Cmd_constant, Mode)

ACTION: Similar to MACRO_TO_KEY except that a Multi-Edit command is assigned to a key instead of a macro name. A complete list of all legal command names is listed in COMMANDS THAT ARE DIRECTLY ASSIGNABLE TO A KEY.

EXAMPLE:

```
  {Assign the command WORD_RIGHT to the
   Control-Right-Arrow key from edit mode}
CMD_TO_KEY( <CtrlRT>, WORD_RIGHT, Edit);
```

UNASSIGN_KEY(Key_code, Mode)

ACTION: Removes any macro, command or keystroke macro assignment from the specified key in the specified Multi-Edit mode.

UNASSIGN_ALL_KEYS

ACTION: Removes any macro, command or keystroke macro assignment from all keys in all Multi-Edit modes.

33. KEYSTROKE MACROS

KEY_IN(keystroke string)

The KEY_IN statement is used to pass keystrokes to editor. The keystroke string is entered without quotes.

EXAMPLE:

```
KEY_IN(This is a test string<ENTER>);
```

would enter in the string followed by a carriage return. Note that currently string variables may not be entered. Keystroke strings may be any number of lines long.

PLAY_KEY_MACRO(keycode1, keycode2)

ACTION:	Starts the specified keystroke macro playing. The keystrokes may then be read via the READ_KEY or CHECK_KEY operations.

KEY_RECORD

ACTION:	Starts or stops the recording of a keystroke macro.

EXTENDED KEY SYNTAX

This is a list of the extended keystroke syntax (this syntax is also used for assigning macros to keystrokes):

<F1>	<ShftF1>	<CtrlF1>	<AltF1>
<F2>	<ShftF2>	<CtrlF2>	<AltF2>
<F3>	<ShftF3>	<CtrlF3>	<AltF3>
<F4>	<ShftF4>	<CtrlF4>	<AltF4>
<F5>	<ShftF5>	<CtrlF5>	<AltF5>
<F6>	<ShftF6>	<CtrlF6>	<AltF6>
<F7>	<ShftF7>	<CtrlF7>	<AltF7>
<F8>	<ShftF8>	<CtrlF8>	<AltF8>
<F9>	<ShftF9>	<CtrlF9>	<AltF9>
<F10>	<ShftF10>	<CtrlF10>	<AltF10>
<ENTER>		<CtrlENTER>	<Alt0>..<Alt9>
<TAB>	<ShftTAB>	<CtrlA>..<CtrlZ>	<AltA>..<AltZ>
<ESC>			
<BS>		<CtrlBS>	<Grey->
<UP>	<ShftUP>		<Grey+>
<DN>	<ShftDN>		<Grey*>
<LF>	<ShftLF>	<CtrlLF>	
<RT>	<ShftRT>	<CtrlRT>	
<PGUP>		<CtrlPGUP>	
<PGDN>		<CtrlPGDN>	
<INS>	<ShftINS>	<<> { less than }	
<DEL>	<ShftDEL>	<>> { greater than }	

<MsUP> { mouse up }
<MsDN> { mouse down }
<MsLF> { mouse left }
<MsRT> { mouse right }

<Btn0> { left mouse button }
<Btn1> { right mouse button }
<Btn2> { middle mouse button }

COMMANDS THAT ARE DIRECTLY ASSIGNABLE TO A KEY

CR
LEFT
RIGHT
UP
DOWN
HOME
EOL
PAGE_UP
PAGE_DOWN
TOF
EOF
NEXT_PAGE_BREAK
LAST_PAGE_BREAK
WORD_LEFT
WORD_RIGHT
MARK_POS
GOTO_MARK

TAB_LEFT
TAB_RIGHT
INDENT
UNDENT
DEL_CHAR
DEL_LINE
BACK_SPACE
UNDO
BLOCK_BEGIN
BLOCK_END
COL_BLOCK_BEGIN
COL_BLOCK_END
COPY_BLOCK
MOVE_BLOCK
DELETE_BLOCK
BLOCK_OFF
SAVE_FILE

MODIFY_WINDOW
UNLINK_WINDOW
CREATE_WINDOW
ENTER_DOS_SHELL
CALCULATOR
QUIT
REDRAW
NEWSCREEN
REST_DOS_SCREEN
SAVE_DOS_SCREEN
KEY_RECORD
PUSH_UNDO
POP_UNDO
DELETE_WINDOW
ERASE_WINDOW
ZOOM
FIRST_WORD

34. MISCELLANEOUS OPERATIONS

BEEP

ACTION: Guess!!

EXAMPLE:

```
   {If an error occurred, then beep}
IF Error_Level <> 0 THEN
   Beep;
END;
```

DELAY(Time)

ACTION: Delays for Time milliseconds.

EXAMPLE:

```
   {Wait for 1/10 second}
Delay( 100 );
```

NOT(integer)

RETURNS: If integer is non-zero, then NOT will return 0.
If integer is zero, then NOT will return 1.
(Used for true/false type expressions)

MODE

RETURNS: An integer containing the current mode or module that is being used.

0 = Text Editing.
1 = DOS Shell.
2 = Communications Module.
3 = (currently undefined)
4 = Calculator.

EXAMPLE:

```
IF Me_Mode = 1 THEN
  EXIT_DOS_SHELL;
END;
```

HELP(Index_str)

ACTION: Brings up the help screen corresponding to the passed 2 character Index_Str. If the index cannot be found, then the main help menu is brought up. (See RETRIEVE_HELP)

EXAMPLE:

```
Help('MA');
```

RETRIEVE_HELP(Index_str, x1, y1, x2, y2, Filename, Title)

ACTION: Brings up a help window whose size is specified by x1, y1, x2 and y2. The help file is specified with Filename, and the help window may be given a Title.

A help file is an ASCII text file formatted so that each subject is separated by a form feed (ASCII 12) character and an index string. There may also be a table of contents section, which is defined with a form feed and an asterisk ('*'). Use the ME.HLP file as an example.

PRINT_STR(String)

ACTION: Sends the string to the PRN device (usually the printer). Get error code via the ERROR_LEVEL function. (See ERROR_HANDLING)

EXAMPLE:

```
Goto_Line(10);
Goto_Col(1);
Print_String(Get_Word(''));
```

VERSION

RETURNS: A string containing the Multi-Edit version number.

PUSH_UNDO

ACTION: Starts an undo record. (see POP_UNDO)

POP_UNDO

ACTION: Finishes an undo record.

EXAMPLE:

```
{this code may be reversed with a single undo command }
PUSH_UNDO;              {start undo}
Del_Chars( 10 );        {delete 10 chars}
Goto_Col( 1 );
Text( 'Far out' );
POP_UNDO;               {stop undo}
```

CALCULATOR

ACTION: Brings up the pop-up calculator

35. SYSTEM VARIABLES

The following are predefined system variables. Many are not actually variables, but complete operations that initiate action, however from the programmers perspective they are used exactly as variables.

STRINGS

RETURN_STR
Used to return string values from a macro. See PARAMETER PASSING AND RETURN VALUES.

FILE_NAME
The name of the file being edited in the current window. May be up to 65 characters long. EXAMPLE:

```
File_Name := 'C:\ME\INIT.SRC';
```

FORMAT_LINE
The format line string for the current file. NOTE that ASCII code 16 is used to represent tab stops in the format line. An 'R' may be imbedded in the line to set the right margin (See the variable RIGHT_MARGIN). No other characters are treated specially. The format line may be up to 254 characters long.

DEFAULT_FORMAT
This is the default format line string. (See FORMAT_LINE)

PAGE_STR
The Page-Break string. Normally set to a single Form Feed (ASCII code 12) character, but may be any string up to 10 characters in length.

WORD_DELIMITS
Defines the word delimiters that are used by WORD_LEFT and WORD_RIGHT.

SHADOW_CHAR
The character used to draw the box shadows.

INTEGERS

RETURN_INT

Used to return integer values from a macro. See PARAMETER PASSING AND RETURN VALUES.

ERROR_LEVEL

An integer containing the error number of the last error that took place (See the section titled ' ERROR HANDLING').

IGNORE_CASE

Set to TRUE (1) to ignore the case of alpha characters during a search. (See SEARCH AND REPLACE OPERATIONS)

REFRESH

This flag will turn the updating of the screen off or on. (1 = On, 0 = Off) Setting REFRESH to off will significantly increase the performance of most macros.

FILE_CHANGED

If this flag is true (1) then the file has been modified since being loaded or saved. Saving a file will set this flag to false (0).

INSERT_MODE

If this flag is true (1) then the editor is in text insert mode, otherwise it is in overwrite mode. Note that assigning a value to this flag will update the Insert/Overwrite display and change the cursor shape.

PRINT_MARGIN

The margin set for printing of a file or block.

MESSAGES

If this flag is false, then the built-in functions of Multi-Edit will NOT display their usual messages on the message line.

MOUSE

Turns recognition of mouse functions on (1) and off (0).

LOGO_SCREEN

Only effective if SAVE_SETUP is invoked. Setting LOGO_SCREEN to 0, then invoking SAVE_SETUP will bypass the logo screen on subsequent Multi-Edit sessions.

EXPLOSIONS

TRUE (1) = Pop up boxes and windows will explode.
FALSE (0) = No explosions.

TRUNCATE_SPACES

TRUE (1) = Multi-Edit will truncate any trailing spaces on a line.
FALSE (0) = don't truncate trailing spaces.

BACKUPS

TRUE (1) = Make backup files.
FALSE(0) = Do not make backup files.

AUTOSAVE

If 0, then automatic file save is off, else AUTOSAVE equals the number of seconds that the computer must be idle before the autosave macro is invoked.

UNDO_STAT

TRUE (1) = Undo is on globally.
FALSE (0) = Undo is off globally.

FORMAT_STAT

TRUE (1) = Save format line with file.
FALSE (0) = Do not save format line with file.

WRAP_STAT

TRUE (1) = Wordwrap is on in the current window.
FALSE (0) = Wordwrap is off in the current window.

DOC_MODE

TRUE (1) = Multi-Edit is in document mode.
FALSE (0) = Simple text edit mode.

MEM_ALLOC

The amount of ram (in k bytes) to leave free when Multi-Edit is started.

RIGHT_MARGIN

The right hand margin for the current file. Used for word wrap.

INDENT_STYLE

The indent type for the current file.
0 = Manual indent.
1 = Automatic indent.
2 = Smart Auto-indent.

INS_CURSOR

The cursor shape for the Insert mode cursor.
0 = Underline
1 = 1/2 block
2 = 2/3 block
3 = Full block

OVR_CURSOR

The cursor shape for the Overwrite mode cursor. Values are the same as INS_CURSOR.

CTRL_HELP

TRUE (1) = Ctrl-keypad help box is on.
FALSE (0) = Ctrl-keypad help box is off.

MOUSE_H_SENSE

The horizontal mouse sensitivity. The higher the number, the less sensitive the mouse will be.

MOUSE_V_SENSE

The vertical mouse sensitivity. The higher the number, the less sensitive the mouse will be.

WINDOW_ATTR

The attributes for the current window. The bit settings are as follows:
Bit 0 = (1) hidden, (0) not hidden.
Bit 1 = (1) status info not displayed, (0) status info displayed.
Bit 2 = (1) border not displayed, (0) border displayed.

REG_EXP_STAT

True (1) = Regular expressions enabled.
True (0) = Regular expressions disabled.

TAB_EXPAND

TRUE (1) = Use tab characters and pad with virtual space.
FALSE (0) = Use space character instead of tabs.

TEXT_COLOR

The color assigned to text being edited (see the list of PREDEFINED CONSTANTS for color values).

CHANGE_COLOR

The color assigned to highlight changed text.

BACK_COLOR

The color assigned to the background.

MENU_COLOR

The color assigned to the unhighlighted portion of a menu.

STAT_COLOR

The color assigned to the highlighted portion of the status line and menus.

ERROR_COLOR

The color assigned to error messages.

SHADOW_COLOR
The color assigned to the box shadows.

STATUS_ROW
The row to display the status line on. 0 = Off.

MESSAGE_ROW
The row to display the message line. 0 = Off.

MAX_WINDOW_ROW
The maximum row that a window may be displayed on.

MIN_WINDOW_ROW
The minimum row that a window may be displayed on.

NAME_LINE
0 = Don't use bottom line of window. Display filename on top. 1 = the opposite.

FKEY_ROW
The row to display the function key labels. 0 = Off.

DISPLAY_TABS
1 = display tab characters. 0 = don't display tab characters.

INIT_VIDEO_MODE
Used strictly by the INIT macro to set the initial video mode. Same values as the function VIDEO_MODE.

EMS_STAT
True = Expanded memory is being used. Once set to true it may NOT be set back to false.

36. ERROR HANDLING

Many operations, particularly file operations, may generate errors (like 'File not found' or 'Device not ready'). These operations set the ERROR_LEVEL variable to an error number (this is strictly a Multi-Edit variable, and has no relation to the DOS ERRORLEVEL batch function). Error number 0 means that no error occurred. NOTE that any operation that sets the ERROR_LEVEL variable has a notation to that effect in it's description.

Whenever an operation that has the potential to generate an error (particularly if the macro is dependant on the success of that operation) is invoked, the ERROR_LEVEL variable should be checked for a non-zero value. If it contains a non-zero value then the macro should process the error in some recoverable manner, and then set the ERROR_LEVEL variable back to zero. If all that is necessary is to display an error message, then the ME-ERROR macro may be run.

EXAMPLE:

```
   {Load a file}
Load_File('C:\ME\INIT.SRC');
   {If an error occurred then run the ME-ERROR macro and
exit}
IF Error_Level <> 0 THEN
   Run_Macro('ME-ERROR');
   Goto EXIT;
END;
```

MULTI-EDIT ERROR CODES

1001	No windows left. CREATE_WINDOW was invoked when 100 Windows already exist.
1002	Insufficient memory to complete operation.
1007	ME.HLP not found. The HELP operation was invoked, but Multi-Edit can't find the ME.HLP file.

1010	Invalid search string. SEARCH_FWD or SEARCH_BWD was invoke with invalid regular expression syntax. The most common problem is unbalanced braces around groups.
1011	Input file not found. A SAVE_FILE was performed while in disk swapping mode, but the original file can't be found. This could happen if the file has been deleted from the DOS shell.
2000	Write protect error. An attempt was made to save a file to a write protected disk.
2002	Device not ready. Attempt was made to output to a device that was not ready or setup.
2003	Unknown command. This is a DOS error. Should never happen.
2004	Data error. A disk is probably damaged.
2008	Sector not found.
2009	Printer out of paper.
2010	DOS Write fault error.
2011	DOS Read fault error.
2012	DOS General failure error.
3002	File not found. LOAD_FILE can generate this error.
3003	Path not found.
3007	DOS Memory Control blocks destroyed.
3008	Insufficient memory. SHELL_TO_DOS can generate this error.
3010	Invalid environment.

3015	Invalid drive. Most people do not have a drive Z.
3021	Drive not ready.
3031	General failure.
3241	Disk full. SAVE_FILE can generate this error.
5001	Macro not found. An attempt was made to load or run a macro, but Multi-Edit could not find the .MAC file.
5005	Macro being loaded is an incompatible version. Will happen if an attempt is made to load a macro that was compiled with an earlier version of the macro compiler.
5006	Macro running cannot be erased or reloaded. Simply put, a macro can't erase or reload itself.

37. RUN-TIME ERRORS

A macro run-time error may be generated under several circumstances. If such an error occurs, a window will appear telling the macro name and program counter (in hex) where the error occurred. This information should immediately be written down, so that the position of the error can be found in the source code.

To find the position of a run-time error you must invoke the compiler from the DOS command line as follows:

```
MEMAC filename -Dpc macro_name
```

EXAMPLE:

```
MEMAC LANGUAGE -D002A TEMPLATE
```

The above example would find a run time error at program counter 002A hex in the macro TEMPLATE which is in the macro source file LANGUAGE.SRC.

RUN-TIME ERROR CODES

01 Attempt to execute command during wrong mode.
Would happen if, for example, the macro attempted an editing operation while in the DOS Shell.

02 Attempt to RET without matching CALL.
Happens when the CALL stack is empty. Could happen if a macro performed a GOTO to a subroutine that terminates with a RET.

04 Real to Integer conversion out of range.
The result of an R_INT function in which the passed real number value was outside of the range of -32768 to 32767.

05 Attempt to nest more than 10 keystroke macros.

Would happen if a Key_In statement invoke a macro which in turn invoked a Key_In statement, and so on, passed ten levels of nesting.

06 Invalid string index on string copy operation.

The string index on a copy operation must be in the range 1 to 254.

07 Corrupt macro code.

Should never happen. If it does, CALL US!

10 String length error.

Happens if the result of a string concatenation has a length greater than 254.

38. *COMPILER ERRORS*

The format for compiler errors is:

```
filename(line num,col num): ERROR errornum: message
```

EXAMPLE:

```
LANGUAGE.SRC(92,22): ERROR 1: Syntax Error: 'THIN'
```

COMPILER ERROR CODES

01	Syntax Error: <statement>.	The compiler does not recognize the displayed statement.
02	<delimiter> expected.	The displayed delimiter (a parenthesis, comma, etc.) was expected but not found.
04	; expected.	A semicolon was expected, but not found.
05	Label to long.	A label with more than 20 characters was defined.
06	Too many labels.	More than 500 labels where defined within a macro.
07	'(' expected.	A left parenthesis was expected.
08	',' expected.	A comma was expected.
09	')' expected.	A right parenthesis was expected.

13 Variable expected.

An actual variable (not a constant, literal or expression) was expected.

14 Label <label name> not found.

A label was referenced but never defined.

15 Type mismatch or syntax error.

The constant, literal, expression or variable is either or the wrong type, or contains a syntax error.

16 Macro name must be <= 8 characters long.

The name of the macro is too long.

17 Out of variables.

Too many local variables have been defined. The maximum number of local variables is 200.

24 Keycode expected.

A proper keycode string was expected. See 'KEYSTROKE MACROS'.

25 Mode expected.

A Multi-Edit mode was expected.

27 END statement not found.

There are not enough end statements to match IF and WHILE statements in the macro.

28 $command expected.

Either a $MACRO or $MACRO_FILE was expected.

29 Out of memory for compiler.

Not enough memory to compile macro. You probably need to exit the editor and compile the macro from the command line.

30 $MACRO_FILE already defined.

Only ONE $MACRO_FILE definition is allowed per macro source file.

31 String constant too long.

A string constant must be <= 254 characters long.

32 Multi-Edit command expected.

Attempt to assign an illegal command to a key. See 'COMMANDS THAT ARE DIRECTLY ASSIGNABLE TO A KEY'.

100 Premature end of file.

The file ended before an END_MACRO statement was encountered.

101 File not found.

Source file was not found.

102 Output file error.

An error occurred writing the output file.

103 Invalid debugging PC.

The program counter specified in the -D command was invalid. See 'RUN-TIME ERRORS'.

200 Runtime error found.

The runtime error specified by -D command was found. 'SEE RUN-TIME ERRORS'.

39. EXAMPLE MACRO

```
{**************************************************
          --- SMART INDENT FOR C ---
 Multi-Edit macro to perform a smart indent each
 time the Carriage Return or Enter key is pressed.
**************************************************}

$MACRO C_IND;
   DEF_STR( C_STR );                     {Stores the word to check for indent}
   DEF_INT( T_COL,                       {Position of first word on line}
            T_COL2 );                    {Original position}
   T_COL2 := C_COL;                      {Store the current position}
   FIRST_WORD;                           {Move cursor to first word on line}
   T_COL := C_COL;                       {Store the position}
   IF T_COL2 < T_COL THEN                {If the cursor was before the 1st word}
      T_COL := T_COL2;                   {Then goto to the original position}
      GOTO_COL(T_COL);                   { So that indenting occurs at that }
      END;                               { position instead of the first word}
                                         {If we are not at the end of the line }
   IF Cur_Char <> '|0' THEN
      SET_INDENT_LEVEL;                  { then set the indent level to the }
   END;                                  { current cursor position }
   T_COL := C_COL;                       {Store the current position}
                                         {Get the first word on the line,
                                          removing any extraneous space}
   C_STR := REMOVE_SPACE( GET_WORD('; (,\') );
   GOTO_COL(T_COL2);                     {Return the cursor to the original
                                         position}
   CR;                                   {Perform a carriage return}
                                         {If the word stored in C_STR matches
                                         any in the list, and the original position
                                         was not at the first character of the first
                                         word, then perform an indent}
  IF (T_COL <> T_COL2) AND LENGTH(C_STR) <> 0) AND
     (POS(C_STR,    ' if while do for { switch ') <> 0) THEN
    INDENT;
  END;
MAC_EXIT:
END_MACRO;
```

Index

.MAC (3), (4)
$MACRO
    definition (2)
$MACRO_FILE (3), (4)
+ (13)
- (13)
* (13)
/ (13)
< (13)
<= (13)
<> (13)
= (13)
> (13)
>= (13)
AND (13)
ASCII (9), (25)
AT_EOF (35)
AT_EOL (35)
AUTOSAVE (82)
BACK_COLOR (84)
BACK_SPACE (31)
BACKUPS (82)
BAR_MENU (49)
BEEP (76)
BLOCK OPERATIONS (39)
BLOCK_BEGIN (39)
BLOCK_COL (41)
BLOCK_END (39)
BLOCK_LINE (40)
BLOCK_OFF (41)
BLOCK_STAT (40)
BOOLEAN (7)
BRANCHING (18)
    CONDITIONAL (19)
    UNCONDITIONAL (18)
BUFFER_ID (57)
C_COL (35)
C_LINE (35)
C_PAGE (66)
C_ROW (35)
CALCULATOR (79)
CALL (18)
CAPS (25)
CHANGE_COLOR (84)
CHANGE_DIR (65)
CHAR (25)
Character (7)
CHECK_KEY (47)
CLEAR_SCREEN (46)
CLR_LINE (43)
CMD_TO_KEY (71)
COL_BLOCK_BEGIN (39)
Command line (4)
COMMAND.COM (61)
COMMANDS (75)
Comments (6)
COMPILING (4)
COMSPEC (61)
Constant
    predefined (10)
Constants (9)
COPY (25)
COPY_BLOCK (39)
CPU (64)
CR (30)
CREATE_WINDOW (52)
CRUNCH_TABS (38)
CTRL_HELP (83)
CUR_CHAR (29)
CUR_WINDOW (54)
Cursor
    POSITION (32)
DATE (62)
DEF_CHAR (8)
DEF_INT (8)
DEF_REAL (8)
DEF_STR (8)
DEFAULT_FORMAT (80)

DEL_CHAR (30)
DEL_CHARS (30)
DEL_FILE (65)
DEL_LINE (31)
DELAY (76)
DELETE_BLOCK (40)
DELETE_WINDOW (53)
DIR (67)
DIR_ENTRY (68)
DIR_MASK (68)
DIR_NUM (69)
DIR_PATH (69)
DIR_TOTAL (69)
DISPLAY_TABS (85)
DO (19)
DOC_MODE (83)
DOS (61)
DOS SHELL (67)
DOS_BACK (64)
DOS_COLOR (64)
DOS_DOWN (67)
DOS_EOF (68)
DOS_FILE_ATTR (70)
DOS_PGDN (68)
DOS_PGUP (68)
DOS_TOF (68)
DOS_UP (67)
DOS_VERSION (64)
DOWN (32)
ELSE (19)
EMS_STAT (85)
END (19)
END_MACRO (2)
ENTER_DOS_SHELL (67)
EOF (32)
EOF_IN_MEM (57)
EOL (32)
ERASE_WINDOW (53)
Error
    CODES (86)
    COMPILER (91)
    HANDLING (86)
    RUN-TIME (89)
ERROR_COLOR (84)
ERROR_LEVEL (81)
EXECUTING A MACRO (5)
EXIT_DOS_SHELL (67)
EXPAND_TABS (38)
EXPLOSIONS (82)
EXPRESSIONS (13)
    integer (13)
    order of evaluation (13)
    real (15)
    string (14)
FILE (56)
FILE_CHANGED (81)
FILE_EXISTS (56)
FILE_MARKED (69)
FILE_NAME (80)
FIRST_FILE (57)
FIRST_GLOBAL (12)
FIRST_MACRO (22)
FIRST_RUN (22)
FIRST_SAVE (57)
FIRST_WORD (33)
FKEY_ROW (85)
FLABEL (51)
FORMAT_LINE (80)
FORMAT_STAT (82)
FROM (2)
FUNCTION KEY (51)
    LABEL (51)
GET_ENVIRONMENT (61)
GET_EXTENSION (27)
GET_LINE (29)
GET_PATH (28)
GET_WORD (29)
Global variables (11)
GLOBAL_INT (11)
GLOBAL_STR (11)
GOTO (18)
GOTO_COL (34)
GOTO_LINE (34)
GOTO_MARK (34)
GOTOXY (44)
HELP (77)

HOME (32)
IF (19)
IGNORE_CASE (81)
INDENT (37)
INDENT_LEVEL (37)
INDENT_STYLE (83)
INDENTING (37)
INIT_VIDEO_MODE (85)
INQ_MACRO (22)
INS_CURSOR (83)
INSERT_MODE (81)
INT_R (16)
Integer (7)
KEY (48)
    ASSIGNMENTS (71)
    EXTENDED (74)
    KEYSTROKE MACROS (73)
KEY_IN (73)
KEY_RECORD (73)
KILL_BOX (42)
Label (6)
LAST_FILE_NAME (58)
LAST_PAGE_BREAK (36)
LEFT (32)
LENGTH (25)
LINK_STAT (54)
LINK_WINDOW (54)
LOAD_FILE (56)
LOAD_MACRO_FILE (21)
Local variables (8)
LOGO_SCREEN (82)
MACRO (21)
MACRO_TO_KEY (71)
MAKE_MESSAGE (43)
MARK_FILE (69)
MARK_POS (33)
MARKING (41)
MAX_WINDOW_ROW (85)
ME_PATH (62)
MEM_ALLOC (83)
MEMAC.EXE (4)
MENU_COLOR (84)
MESSAGE_ROW (85)
MESSAGES (81)
MIN_WINDOW_ROW (85)
MOD (13)
MODE (77)
MODIFY_WINDOW (53)
MOUSE (82)
MOUSE_H_SENSE (84)
MOUSE_V_SENSE (84)
MOVE_BLOCK (39)
MPARM_STR (23)
NAME_LINE (85)
NEW_SCREEN (54)
NEXT_FILE (58)
NEXT_GLOBAL (12)
NEXT_MACRO (22)
NEXT_PAGE_BREAK (36)
NOT (76)
NUL (7)
OR (13)
OVR_CURSOR (83)
PAGE_DOWN (34)
PAGE_STR (80)
PAGE_UP (34)
PARAM_COUNT (63)
PARAM_STR (63)
PASS_KEY (48)
PG_LINE (66)
PLAY_KEY_MACRO (73)
POP_LABELS (51)
POP_MARK (34)
POP_UNDO (79)
POS (26)
PRINT_MARGIN (81)
PRINT_STR (78)
PUSH_KEY (48)
PUSH_LABELS (51)
PUSH_UNDO (79)
PUT_BOX (42)
PUT_COL_NUM (45)
PUT_LINE (29)
PUT_LINE_NUM (45)
QUIT (63)

READ_KEY (47)
Real (7)
REAL_I (16)
REDRAW (54)
REFRESH (81)
REG_EXP_STAT (84)
REMOVE_SPACE (27)
REPLACE (59), (60)
REST_DOS_SCREEN (61)
RET (18)
RETRIEVE_HELP (78)
RETURN_INT (81)
RETURN_STR (80)
RIGHT (33)
RIGHT_MARGIN (83)
RSTR (16)
RUN_MACRO (5), (21)
RVAL (17)
SAVE_DOS_SCREEN (62)
SAVE_FILE (56)
SCREEN (42)
SCREEN_LENGTH (44)
SCREEN_WIDTH (44)
SCROLL_BOX_DN (46)
SCROLL_BOX_UP (46)
SEARCH (59)
SEARCH_BWD (59)
SEARCH_FWD (59)
SET_GLOBAL_INT (12)
SET_GLOBAL_STR (12)
SET_INDENT_LEVEL (37)
SHADOW_CHAR (80)
SHADOW_COLOR (85)
SHELL_TO_DOS (61)
SHL (13)
SHR (13)
SIZE_WINDOW (52)
STAT_COLOR (84)
STATUS_ROW (85)
STR (27)
STR_BLOCK_BEGIN (39)
STR_DEL (26)
STR_INS (27)
String (7)
    MANIPULATION (25)
STRING_IN (50)
SWITCH_WINDOW (52)
TAB_EXPAND (84)
TAB_LEFT (37)
TAB_RIGHT (37)
TABS_TO_SPACES (38)
TABSTOPS (37)
TEXT (30)
    MANIPULATION (29)
TEXT_COLOR (84)
THEN (19)
TIME (62)
TMP_FILE (57)
TMP_FILE_NAME (57)
TO (2)
TOF (32)
TOGGLE_VIDEO (44)
TRANS (2)
TRUNCATE_EXTENSION (28)
TRUNCATE_PATH (28)
TRUNCATE_SPACES (82)
Type checking (7)
Types (7)
UNASSIGN_ALL_KEYS (72)
UNASSIGN_KEY (72)
UNDENT (37)
UNDO_STAT (82)
UNLINK_WINDOW (54)
UNLOAD_MACRO (22)
UP (32)
UPDATE_DIR (68)
USER INPUT (47)
V_MENU (50)
VAL (27)
Variables
    SYSTEM (80)
VERSION (78)
VIDEO_CARD (45)
VIDEO_MODE (45)
Virtual characters (38)
WHEREX (44)

WHEREY (44)
WHILE (19)
WIN_X (55)
WIN_Y (55)
WINDOW (52)
WINDOW_ATTR (84)
WINDOW_COPY (40)
WINDOW_COUNT (55)
WINDOW_MOVE (40)
WORD PROCESSING (66)
WORD_DELIMITS (80)
WORD_LEFT (33)
WORD_RIGHT (33)
WORD_WRAP_LINE (66)
WORKING (45)
WRAP_STAT (83)
WRITE (43)
WRITE_SOD (65)
XPOS (26)
ZOOM (54)